RMIT UNIVERSITY Computer Science and Information Technology

RMIT School of Computer Science and Information Technology

# Programming 2

## Topic 2: Inheritance and Polymorphism

Lecture Slides



## Inheritance: the Credit Account Example

| Account |
| --- |
| acctNumber : int<br>balance : double |
| withdraw(amount : double) : boolean<br>deposit(amount : double)<br>getAcctNumber() : integer<br>getBalance() : double |

The simple Account class from previously

The class has:

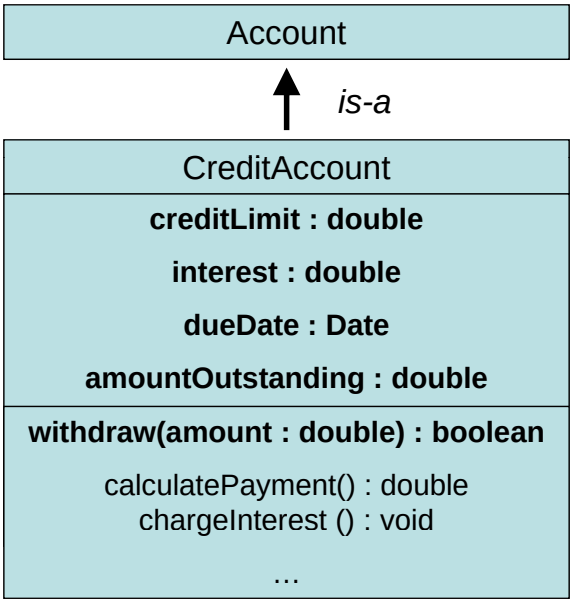
A *Name* (should be unique and descriptive)

*Attributes* (also called fields which are variables representing the **state** of the class)

*Methods* (which are functions representing the **behaviour** of the class)

*Relationships* to other classes represented as attributes (which may be collections for 'many' relationships)

## Inheritance: the Credit Account Example




The bank introduces a new product, the credit account. This has a credit limit, an interest rate, a payment due date, and a payment amount outstanding.

A customer may withdraw up to the credit limit minus the account balance.

Every month, a credit charge based on the outstanding amount is be charged to the account.

Inheritance promotes **reusability** (code reuse) and **extensibility** (extending or modifying existing code).

Every object in Java, either explicitly or implicitly, is a subclass of `Object`.



## The CreditAccount class: `CreditAccount.java` (1)

```
class CreditAccount extends Account {

  private double creditLimit = 0.0F;
  public double interest = 0.0F;
  private Date dueDate = new Date();
  private double outstanding = 0.0;

  public CreditAccount(int num, double creditLimit,
                       double interest)
  {
    super(num, 0); // set the balance to 0 for new account
    this.creditLimit = creditLimit;
    this.interest = interest;
  }

  public void deposit(double amount){
     setBalance(getBalance() - amount);
     outstanding -= amount;
     if (outstanding < 0)
       outstanding = 0;
  }
```

Inherits from `Account`

Calling the superclass constructor

**Overriding the deposit method inherited from Account**

## The CreditAccount class: `CreditAccount.java` (2)

```
public boolean withdraw(double amount){
  if (creditLimit - getBalance() >= amount)
  {
    setBalance(getBalance() + amount);
    return true;
  }
  else
    return false;
}

public void chargeInterest(){
  setBalance(getBalance()+(getBalance() * (interest / 12)));
}
```

Overriding the withdraw method inherited from `Account`

A new method, `chargeInterest`



## The CreditAccount class: `CreditAccount.java` (3)

```
public double calculateOutstanding(){
  if (getBalance() > 0)
      return getBalance() / interest; // not nec realistic!
  else
      return 0.0;
}
}
```

A new method, `calculateOutstanding`

In summary, the `CreditAccount` class:

- inherits properties from the `Account` class
- calls the superclass constructor to help create itself
- overrides some methods with a new implementation
- defines some new attributes and methods unique to `CreditAccount`

## The super and this Keywords

A subclass may explicitly access a method or attribute in its superclass with the `super` keyword. When `super` is used to access the constructor of the superclass, it must be the first statement in the subclass' constructor.

The `this` keyword refers to the object through which the method or attribute is accessed (the current object.) In the absence of an object reference (or class name, for static items), this is the implied reference.

```
public CreditAccount(int num, double creditLimit,
                     double interest)
{
  super(num, 0); // set the balance to 0 for new account
  this.creditLimit = creditLimit; // this resolves the
  this.interest = interest;       // ambiguity
   balance=0; // this is not necessary since no ambiguity
}
```



## Polymorphism and Dynamic Binding (1)

When the bank manager asks his/her staff to deduct $5.00 from every bank account, the staff know that different deduction methods will apply to different types of account. Hence the action deduction is polymorphic.

When the method `withdraw()` is called on an `Account` object, depending on the actual type of that `Account` at the time, the correct version of `withdraw()` will be dynamically bound to that object and executed.

First, let's add several accounts of different types for a customer:

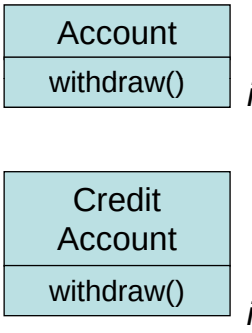
```
Customer john = new Customer("John", 123);
john.addAccount(new Account(1, 500));
john.addAccount(new CreditAccount(2, 0, 200));
```

Recall that the method `addAccount()` of `Customer` accepts an `Account` type as a parameter. Why can it also take in a `CreditAccount` type?

This is because `CreditAccount` "is-a" `Account`, due to inheritance.

## Polymorphism and Dynamic Binding (2)

Now, to withdraw $5.00 from all of John's accounts. Which version of `withdraw()` is executed when?




```
Customer john = new Customer("John", 123);
john.addAccount(new Account(1, 500));
john.addAccount(new CreditAccount(2, 0, 200));

for (int i = 0; i < 2; i++) {

  Account a = john.getAccount(i);
  a.withdraw(5);
}
```

*i=0*

*i=1*

We can put different types of `Account` in a common `Account` array, and withdrawal can be performed polymorphically on them.

`Account` is the base class of all account subclasses. Polymorphism promotes code reuse by allowing methods to be called in a generic way.



## Type Conversion

So, a `CreditAccount` can be treated as an `Account`. However, the reverse is not true; an `Account` may not be a `CreditAccount`.

```
Account ac;
CreditAccount creditAc;
ac = creditAc;
creditAc = ac;
```

Can we convert an `Account` to a `CreditAccount`? Yes, by **type casting**; this can be done only if the `Account` is actually a `CreditAccount`, otherwise a `ClassCastException` will be thrown.

```
creditAc = (CreditAccount) ac;
```

*In summary:*

- a subclass reference is automatically a superclass reference, but the reverse is not true.
- It is possible to convert a superclass reference to a subclass reference if the former is actually referencing a subclass object, otherwise Java will throw a `ClassCastException`.
- Avoid casting and use polymorphism to handle type differentiation

## The `instanceof` Operator (1)

What if we call `chargeInterest()` on an `Account` object?

```
Customer john = new Customer("John", 123);
john.addAccount(new Account(1, 500));
john.addAccount(new CreditAccount(2, 0, 200));

for (int i = 0; i < 2; i++) {
  Account a = john.getAccount(i);
  a.withdraw(5);
  a.chargeInterest();
}
```

We know that we can call `chargeInterest()` on a `CreditAccount` object.

We also know that a superclass reference can be cast to a subclass reference, if it is actually holding a suitable subclass object.

The `instanceof` operator can perform a check to ensure this is the case.



## The `instanceof` Operator (2)

```
Customer john = new Customer("John", 123);
john.addAccount(new Account(1, 500));
john.addAccount(new CreditAccount(2, 0, 200));

for (int i = 0; i < 2; i++) {
  Account a = john.getAccount(i);
  a.withdraw(5);
  if (a instanceof CreditAccount)
     ((CreditAccount) a).chargeInterest();
}
```

Checks if a is storing a `CreditAccount` object

An inline cast – a one-off shortcut

- **In general this technique should be avoided and polymorphism used instead**
- **Can use distinct collections and perform separate operations on them**
- **Characteristics unique to a class should be embedded within that class**

## The `instanceof` Operator (3)

In summary, the `instanceof` operator is called in the following manner:

**`<object reference> instanceof <class name>`**

This will check whether or not the object referenced by <*object reference*> is an instance of class <*class name*>.



## Additional Lecture Resources

PowerPoint slides from chapter 9 Inheritance and Polymorphism of the prescribed textbook *Introduction to Java Programming by Y. Daniel Liang, Prentice-Hall, 2007*, are available on Blackboard and will be presented during the lecture. You should study the book chapter and slides and may wish to print the slides so that you have a hard copy during the lecture.